Interactive Visualization of High-Velocity Event Streams

Uwe Jugel
SAP Research Dresden
Chemnitzer Str. 48
01187 Dresden
uwe.jugel@sap.com

Volker Markl*
Technische Universität Berlin
Straße des 17. Juni 135
10623 Berlin
volker.markl@tu-berlin.de

ABSTRACT

Today, complex event processing systems enable real-time analysis of high-velocity event streams. Considering their efficiency for high-speed data analytics, they provide a promising basis for real-time visualization. However, a CEP system has to deal with several streaming-specific problems when being used for modern, web-based visualizations. Such visualizations do not only consume streaming data in real-time, but should also provide advanced, interactive exploration options, run on mobile devices, and scale efficiently for mass user applications.

In this paper, I define three core challenges for CEP systems regarding interactive, real-time visualization for future web applications. Within my PhD work, I want to meet those challenges by investigating (1) *Interactivity Operators*, solving problems with long running queries, (2) backend-powered *Visualization Operators*, relieving mobile devices of rendering duties, and (3) *Multi-User Visualization Pipelines* that avoid redundant data processing when serving visualizations to thousands of event stream consumers.

1. MOTIVATION

Complex Event Processing (CEP) [21] is an established technology for processing high-velocity data in real-time, for scenarios where sub-second latency matters significantly. The most common applications are electronic trading systems. Other scenarios, where real-time processing of events is crucial, are military surveillance, manufacturing automation, pipeline monitoring, fraud detection, and tolling [10].

CEP systems can process data rapidly and provide the resulting *Complex Events* to higher level applications. In many cases, a human being will be informed by the system on certain complex events, and often the system continuously pushes high-frequency data to many connected clients. All data may be further processed and will eventually be visualized on a client device, for example, as simple bitmap or

*Volker Markl is Uwe's PhD supervisor.

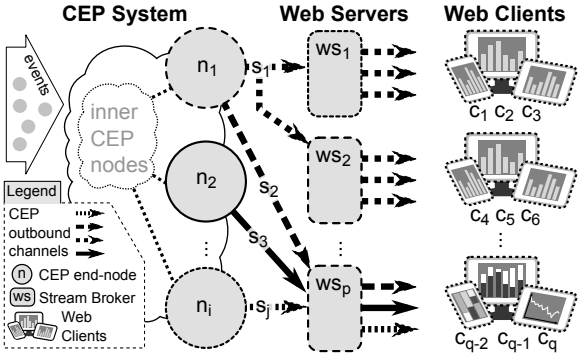


Figure 1: CEP for web-scale, real-time visualization

– now more often – as interactive chart in a visual analytics application. Good business intelligence tools follow the *Visual Analytics Mantra* by Keim et al. [8, p. 83]:

> Analyze First, Show the Important, Zoom, Filter and Analyze Further, Details-on-Demand.

The mantra matches well with the tasks of a CEP engine: *Analyze* and *Filter*. But current CEP systems do not play well with *Zoom*, *Analyze Further,* and *Details-on-Demand*, i.e., the need for interactivity. With a growing number of users and devices, and a growing number of use cases, new requirements are put on the CEP system and the entire stream processing and visualization pipeline to better support interactive, real-time visualizations.

Therefore, in my PhD thesis, I will define, develop, and evaluate a processing pipeline that meets these requirements and allows for web-scale processing and distribution of event streams to visualize them on any modern web client, such as mobile phones and tablets. Figure 1 depicts a high level view for the envisioned system, leveraging the power of a distributed *CEP System* that can spread workload on-demand to many CEP nodes, and combine it with efficient *Web Server* technology, working as event stream broker and visualization pipeline for the connected web clients. I want to analyze and optimize the event processing inside this whole system, especially considering the requirements and potential constraints for continuous rendering of event streams in web-browsers on mobile devices.

The remainder of this paper is structured as follows. First, I introduce the basic optimization concepts of CEP in Section 2. Then, in Section 3, I describe the current challenges of interactive event stream visualization to establish three

core problems that I want to investigate in my PhD work. In Section 4, I describe my basic concepts to solve these problems, following up with Section 5, proposing two architectures for implementing these concepts. After an overview of related work in Section 6, I conclude the thesis discussion and provide an outlook on my future work in Section 7.

2. COMPLEX EVENT PROCESSING

CEP is a common approach to handle streaming data scenarios, where thousands to millions of events per second [5] are continuously analyzed to deduce further actions and make decisions. A CEP engine performs the necessary data mining operations using *Standing Queries* on the stream. CEP engines are heavily optimized for this purpose and can often deal with complex data mining tasks while still guaranteeing low latencies.

Many CEP engines can spread processing work across a (virtual) server infrastructure to handle the high work load that is common for such systems. To use any resources efficiently, CEP systems analyze, optimize, and split up user queries into different sub tasks. In particular, all queries are transformed to a network of stateful or stateless *CEP Operators*, i.e., an acyclic graph of operators based on a subset of the *Relational Algebra* for databases in combination with *Temporal Operators* [7].

First, the CEP system applies a *Query Optimization* [15], analyzing the operator graph, e.g., to detect overlap in operator sub-graphs. Thereafter, the system estimates the costs for each operator. Costs are computed in different models, usually incorporating usage of CPU, RAM, and network bandwidth. The overall optimization goal is to maximize throughput and minimize infrastructure cost. The costs are used to define an optimal *Operator Placement*, i.e., a suitable configuration for executing the operators in parallel threads, or different instances of the CEP engine.

3. CHALLENGES

When visualizing event streams, several conceptual and technological challenges arise that are not well covered by current CEP systems and event stream visualization pipelines. The following sections cover these challenges in detail.

3.1 Adding Interactivity to CEP Systems

Users do not only want instant notifications on predefined events, resulting from long running queries, but also interactively drill down into the streamed data and instantly receive the requested information. This often introduces a completely new query to the system. To instantly answer this query, the system needs to store a certain amount of data, based on the exploration options provided to the user. In the worst case, this data comprises *all data* for the *full time span*. Storing a lot of data can improve interactivity, but contradicts to common CEP setups, where millions of events per second stream into the system. Such vast amounts of data do not easily fit into common databases.

Instant Queries. Unfortunately, CEP engines do not yet support serving instant data in response to new queries, since many queries are long running queries, gathering data over certain time windows, and the engine may also remove unqueried data from the processing pipeline to optimize throughput and consumption of system resources. If a new query requests data that is not inside the time windows of running operators, new operators will be added to the pipeline, and the user will not get valid results until the requested time window is filled up again.

To solve this problem, and eventually facilitating the high level of interactivity needed for modern real-time visualizations, I want to define a custom set of *Interactivity Operators* and optimize their processing and scheduling, as described in Section 4.1.

3.2 Streaming Events to Any Device

Today, feature-rich analytics applications are expected to be available on mobile devices. Mobile market places provide an increasing number of such business applications, e.g., RoamBI (roambi.com) and StockTouch (stocktouch.com). When developing such applications, additional requirements have to be considered, regarding device capabilities and underlying technologies.

Device Capabilities. Mobile devices have limited CPU power and memory, and often suffer from limited network connectivity. Therefore, application developers should be very conservative regarding use of these resources to achieve reasonable performance and still save battery power.

Portable Front-Ends. Developing an application for each mobile platform is expensive. The best level of technological reuse is expected from browser-based approaches, i.e., using HTML5 [18]. However, browser-based development imposes additional constraints on achievable performance, data transfer techniques, and security. For instance, all connections have to be established via HTTP, all communication has to respect cross-domain policies, and continuous visualizations have to respect how the browser renders web pages, e.g., in active and inactive tabs.

The whole event processing system has to handle these requirements efficiently, i.e., it must be able to serve the correct, real-time streaming data to many users, using many web browsers, running on many kinds of devices, requesting many views on the same data, without unnecessarily exhausting the device's resources. Current CEP systems do not cope with all of these tasks and leave them to be solved by higher level custom built applications, even though – due to their optimized stream processing engines – they hold the potential to solve them very efficiently.

In my PhD work I want to shift specific visualization-related tasks from the devices to the backend, using a custom, optimized set of *Visualization Operators*, potentially running inside the CEP engine, as described in Section 4.2.

3.3 Serving Visualizations to Many Users

Today, CEP is also leveraged for mass user scenarios. For example, one of SAP's customers estimates 22.000 simultaneous users with 5 queries per user running on a CEP system. Similar numbers can be expected from other scenarios, such as massively multi-player online games [17] and financial applications [5]. In the future, with the growing popularity and performance of HTML5, I expect the front-ends of such applications to be mainly browser-based solutions. The more consumers connect to such a web-scale system the more often users will consume similar or even exactly the same data. Any two processing nodes that compute the same value twice for two consumers are wasting resources. A web-scale event distribution system has to effi-

ciently manage such processing pipelines, for example when converting the incoming complex events to JSON messages.

For static content, such as archived images and videos on the web, huge efforts are undertaken, e.g., by Akamai Technologies (akamai.com), to serve data in time and with lowest, server-side resource consumption. Nevertheless, with the web shifting towards a more real-time, event-driven system, an efficient processing and distribution of dynamic, real-time event streams has to be considered.

Therefore, I want to investigate the underlying data queries and visualization operators, and develop a multi-user processing model to facilitate reuse of processing pipelines in the backend, as described in Section 4.3.

3.4 Hypothesis and Research Questions

The described gaps and real-time visualization challenges, regarding interactivity of CEP systems, device limitations, and multi-user management, converge in three research questions, forming the foundation of my thesis:

Problem 1. Interactive Complex Event Processing: How to define, place, and schedule operators in a distributed CEP system for processing numerous changing queries and providing instant responses?

Problem 2. Efficient Visualization Processing: How to better/further process events in the CEP engine, to facilitate a more efficient rendering of continuous visualizations, especially on devices with limited capabilities?

Problem 3. Efficient Multi-User Visualization Processing: How to efficiently process events to create visualizations for a large number of connected visualization front-ends?

4. SOLUTION CONCEPTS

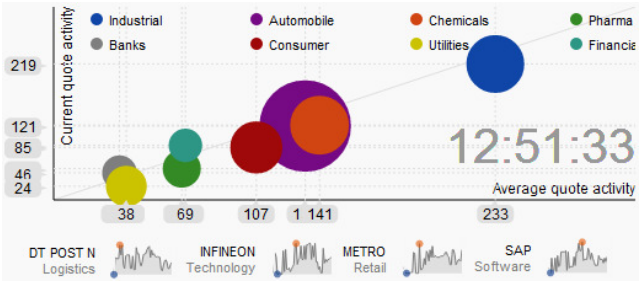
Based on these problems, my goal is to develop a CEP system that can efficiently serve data for real-time visualizations. Therefore, I plan to use a distributed CEP engine as core processing unit, and extend it to efficiently process special *Interactivity* and *Visualization Operators*, facilitating instant interaction with real-time visualizations. The system must handle high-velocity data, high query load, and a high number of simultaneous users. The following sections describe solution approaches for each of the three problems.

4.1 Interactivity Operators

To solve the *Instant Query* problem, and to provide better interactivity, I want real-time visualizations to report their configuration, e.g., the supported range of values, zoom levels, etc., to the CEP system, such that a set of *Interactivity Operators* can be derived and scheduled to the CEP pipeline.

One of the visualizations for our current project comprises a scatter plot and several line charts for presenting the current activity of the stock market (see Figure 2). In this example we stream real-time stock prices and quote data to a (mobile) HTML5 application for visualization and optional drill down. The scatter plot shows the number of quotes in a certain market sector (prime) and allows the user to drill down into the appropriate sub sectors (sub prime). The line charts show stock price trends for the top k companies.

The scatter plot presents aggregations of the data (P and B) and allows de-aggregation on-demand (S and A). One of resulting data management problems is to decide where to



Quotes, prime, sub prime:	$R(q,p,s)$
Sum of quotes in sub market:	$S(s,p,qs) = \pi_{s,p,s} G_{Sum(q)}(R)$
Sum of quotes in market:	$P(p,qp) = \pi_{p,p} G_{Sum(qs)}(S)$
Avg. quotes in sub market:	$A(s,p,qs) = \pi_{s,p,s} G_{Avg(q)}(R)$
Avg. quotes in market:	$B(p,qp) = \pi_{p,p} G_{Avg(qs)}(A)$
Stock price and market cap.:	$M(v,c)$
Price, mcap. for top k stocks:	$T(v,c) = \sigma_{\varphi_k}(\tau_c(M))$

τ_c sorts stocks by market cap. c, and φ limits result to k tuples.

Figure 2: Real-time stock market data visualization with related data and data operators.

process the aggregates. The sub prime aggregation is computed by the backend and the clients compute the primes based on the sub prime values. Backend and front-end process these operations for every tick. The client-side aggregation has to be done for each client, redundantly yielding the same results. When considering the system as a whole, such inefficiencies could be eliminated by doing the aggregate in the backend and using a smarter visualization that receives only the data it currently displays (either primes or sub primes). This could reduce the overall workload and save network traffic.

The current visualization only receives the currently aggregated data for the last timestamp. If the user wants to see past quotes or stock values, the client-side application can only resort on data it already received from the backend. However, a good analytics application would provide a more fine-grained and generic interaction model. Therefore the visualization system has to support two main features.

First, it has to intelligently manage several data stores, i.e., history caches, for each interaction option, thus the clients can request historical data on demand.

Second, the system needs to schedule several visualization-related queries, even if no client currently receives the results. Running these *Shadow Queries* will allow for instant responses for any supported user interaction, i.e., any supported query. For example, the user may want to request stock prices for non-top-k companies.

In a nutshell: to solve thesis *Problem 1*, I need to define rules and strategies for an interactive, real-time visualization system to manage (1) where and how much (historical) data to store and (2) where to process operators on this data.

4.2 Visualization Operators

For supporting efficient rendering, the envisioned system will provide *Visualization Operators* running in the backend, either as part of the application level processing pipeline or as operators in the CEP engine, taking over tasks from any connected client. These clients can be relieved from several visualization pre-processing steps, and will only have to conduct the final rendering.

For example, in our stock market application, in addition to the value aggregation, each client also continuously computes the scatter plot geometry for primes and sub primes, and several line chart geometries for each of the top k stocks. Such geometry computations can make up a significant part of the client-side continuous processing and are therefore good candidates for being outsourced to the backend.

The scalar transformation of quote data to scatter plot coordinates is an example for a *Visualization Operator*. Although, such a scalar transformation of set-based data is actually a very cheap operation that may well run on the client, more expensive operators are those that require to iterate (recursively) over a whole data set to derive a "stable" geometry. Most layout computations belong to this category, e.g., treemaps [16], circle packing [20], and graphs.

To get a better view on the continuous rendering performance of such geometric transformations, I implemented several real-time visualization test cases. For each visualization I separated the layout and geometry computation, i.e., the *Visualization Operator*, from the DOM updating and drawing part of the client-side processing. The visualizations are implemented using D3 [2] (mbostock.github.com/d3), which is reasonably fast and compatible with mobile devices. Figure 3 depicts the results for three different tests: (1) generation of an `svg:path` for a *Line Chart*, (2) *Treemap* layouting, and (3) *Circle Pack* layouting.

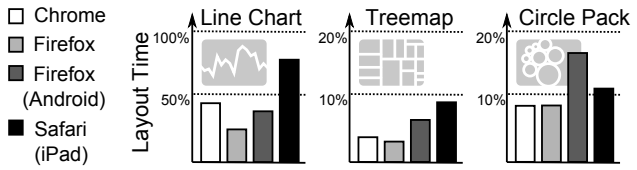


Figure 3: Geometry/layout computation time

The results show that layout computation performance varies a lot with the use case and must not be neglected when building real-time visualizations. Especially for the line chart, I observed a very high workload of up to 78% of the overall client-side processing, which is caused by the numerous string concatenations in the browser's JavaScript VM (i.e., a lot of `memcpy` operations) that are required for creating the `d` attribute of the `svg:path`.

This demonstrates that, depending on the rendering technology, there exist very expensive visualization operators that may heavily benefit from backend processing. Therefore, one goal of my PhD thesis is to identify and classify a common set of such geometric operators and implement them as backend *Visualization Operators* that can enrich event-streams with geometry data.

The implementation requires splitting up geometric transformations into separate steps, as done for the performance tests. In general, I need to analyze the actual continuous data flow processing for different visualizations, i.e., how data sets are transformed to the common geometric primitives. The expected outcome is a clear separation of which parts of the transformation can be expressed as classical relational database operator and which parts require additional processing, e.g., using custom, stateful operators.

My research will facilitate using backend and front-end in an optimal way to speed up the entire real-time visualization pipeline, thereby effectively solving *Problem 2* of my thesis.

4.3 Multi-User Visualization Processing

In Section 1, going back to Figure 1, I already introduced the basic model for event stream distribution on top of a *CEP System*, using a distributed *CEP Engine* and scalable stream-processing *Web Servers* for processing visualizations and pushing events to a large number of *Web Clients* connected to the system.

Figure 1 also shows in detail that multiple users may receive data from a single data source. For example, the clients c_1 to c_6 receive a single event stream s_1 through web servers ws_1 and ws_2. In this case the system is continuously pushing the same data to these clients and has to process it accordingly. This processing includes all running *Interactivity Operators*, but also the *Visualization Operators* and additional tasks, such as the deserialization of binary event data and the generation of web-friendly JSON data.

In a multi-user system many users may require different visualization geometries, e.g., for different screen sizes. However, I expect them to use a common core geometry that could be processed by the backend, leaving the client to conduct only a less costly (e.g., scalar) transformation operation. By analyzing common visual transformations I plan to categorize such visualization operators, and estimate their reuse potential in multi-user scenarios.

This multi-user visualization processing is closely related to the multi-query optimization problem [15]. Therefore, to achieve an optimal processing, I consider reusing, e.g., the CEP engine's query subsumption detection for finding "similar visualizations". Leveraging these subsumptions will reduce the number of processing pipelines per visualization and facilitate reuse of running operators, thereby effectively solving *Problem 3*.

5. TOWARDS INTERACTIVE CEP

In this section, I briefly describe the project context for my PhD work, followed by Sections 5.1 and 5.2, where I present two complementing architectures for solving the three challenges of real-time event stream visualization, as described in Sections 3 and 4.

In our current working project, we are dealing with high-frequency, financial, energy, and manufacturing data, pushed to different web-browsers, running on many kinds of devices, such as iPad, iPhone, and Desktop PCs. We have set up a distributed CEP system, and combined it with a stream-based web application stack to conduct our research.

All data is pushed to the clients solely via WebSockets [19], since we primarily see modern web applications as our future front-ends. The final rendering is done on the devices themselves to provide best interactivity, i.e., the backend will not generate full-sized bitmap images.

5.1 Web-Server-centric Architecture

The first version of an architecture for real-time event stream visualization is depicted in Figure 4. In the *Stream Broker*, the system implements a simple channel-reuse model to avoid redundant data processing. Every time a new query is issued by a client, the query is either sent to the CEP system's *Query Manager*, resulting in a new channel for that query, or, if the query matches with an existing query, the client can be mapped to an existing channel. In the latter case the client instantly receives the correct data, while this may be delayed for completely new user queries, because of the *Instant Query* problem (see Section 3.1).

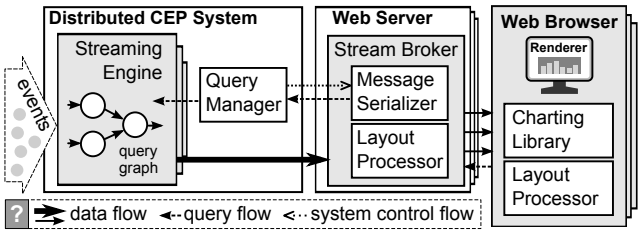


Figure 4: Web-server-centric architecture

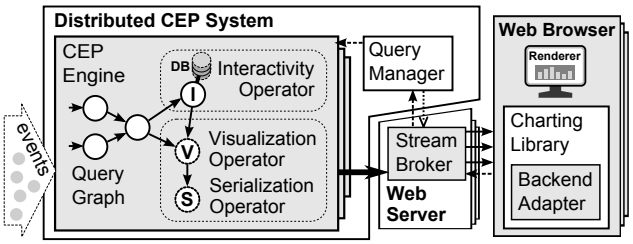


Figure 5: CEP-centric architecture

For mass client support, many instances of web servers can connect to the CEP system, and issue queries of their connected clients. Optimally, the different stream brokers on the different web servers would exchange information about the client connections, to optimize channel reuse across the cluster, e.g., when the same query is sent over two different web servers. Currently, the web servers will execute two separate pipelines, for adding layout information and serializing the event stream. This results in redundant work for the *Message Serializers* and the *Layout Processors*.

In the future, the web servers should be enhanced to act as really distributed streaming web servers, avoiding any redundant processing, e.g., by running the layout processing and serialization in a separate component, managed by the stream broker, and accessible by different web servers.

Our system will allow many users issuing new, arbitrary queries at any time. Therefore, optimizing the visualization processing pipeline is crucial, since the number of running operators in the system not only depends on the requested data, but may grow significantly when considering the work of the *Layout Processor*. For example, with the current system, every backend-computed geometry has to be processed separately if the screen parameters do not match exactly. In this case the system schedules a visualization operator for every screen size, even if they run on the same underlying event stream.

In this architecture, visualizations are processed in a separate component that can not directly benefit from the optimizations that a CEP engine could provide for such kind of event processing. Considering the CEP engine as native visualization processor can solve this problem.

5.2 CEP-centric Architecture

I expect that *Visualization Operators* can be applied to many kinds of event streams, making them good candidates for running inside a CEP engine, where they can be optimized together with the other operators in the pipeline. Leveraging this factor, the web-server-centric architecture can be advanced towards a CEP-centric architecture, depicted in Figure 5, where all visualization-specific operators can be processed directly in the CEP engine. This makes the *Web Server* a pure *Stream Broker* for distributing the event streams and managing HTTP sessions. In addition to visualization processing, events streams have to be serialized, e.g., to JSON or XML, to be consumable by the clients. Such *Serialization Operators* are equally well suited to be processed by the CEP engine, as they usually are *stateless* operations, just as many *Visualization Operators*.

To support interactivity and ensure real-time responsiveness the CEP engine's *Query Manager* must be able to handle the second class of *stateful Interactivity Operators*, as described in Section 4.1, storing history efficiently and using an optimized scheduling model for inactive operators. They must not be removed from the global *Query Graph*, since their data may be requested by the user at any time.

Since the two classes of operators, *Interactivity* and *Visualization Operators*, are derived from visualization requirements of higher level applications, I expect them to exist very close to the leaves of the operator network, i.e., at the end of the *Query Graph*, reflecting the behavior of the external processing pipeline in the web-server-centric system. I want to consider this fact in my PhD work, investigating how local optimizations for these operators can be leveraged and how local and global query optimizations can be performed together. Running *Interactivity* and *Visualization Operators* in a CEP engine in an optimal way will allow the CEP system to better react to changing user queries.

6. RELATED WORK

In the context of real-time visualization on top of a CEP system many different approaches and concepts can be investigated. There are already products and techniques for implementing "real-time" visualizations and there is also significant research on operators, continuous queries, and how to optimize their processing in CEP engines.

Real-Time Visualization Software. Event stream visualizations for web applications are typically developed as custom solutions, e.g., using Java Applets or Adobe Flash [1]; legacy technologies, not compatible with the open web. Another approach is to use dedicated visualization software, such as the solutions by Panopticon (panopticon.com), or specialized toolkits for time series data, such as Graphite (graphite.wikidot.com), which mainly supports bitmap-based rendering. So far, my investigation did not reveal a suitable real-time visualization toolkit that supports interactivity, runs on mobile web-browsers, and scales well with a growing number of clients.

Using Databases with CEP. Databases are used for pseudo-real-time, interactive applications. They can be combined with a streaming system, e.g., by constantly writing events from the CEP system to the database. Any visualizations work directly on top of the database, allowing reuse of existing database-centric analytics tools. This combination of databases and CEP has reached the market in form of products, such as StreamBase LiveView (streambase.com/products/liveview). This model can be further enhanced using in-memory database technology [13] with state-of-the-art CEP engines, such as the Sybase ESP [12]. However, these solutions have some drawbacks, as they have to continuously query the database instead of using push-based techniques. In addition, I also did not find the no-

tion of *Visualization Operators* in the context of databases, even though I expect they could be implemented on top of database technology, such as materialized views.

Visualization Operators. Regarding potential operators and functions for visualization systems, Chi and Riedl [3] provide an in-depth analysis, separating them into view and value operators. The work is built on previous operator research, such as a data flow model for scientific visualization by Schroeder et al. [14] or a visual exploration pipeline for databases by Lee et al. [11]. These works show that the visualization pipeline can be split up into several steps using different operators at different stages, which fits well with the concept of outsourcing operators for visualization processing. The operator classification may also help with preparing visualization operators for direct execution inside a CEP engine. What these approaches are missing is the notion of interactivity. Interactivity relates to the *Instant Query* problem, which I also plan to compare with the problems solved by *Operator Contracts*, as proposed by Childs et al. [4]. They introduce a distributed visualization pipeline with several operators sharing up-stream contract channels, which, according to the authors, "allow all possible optimizations to be applied to each pipeline execution". I consider these works on stream processing operators very relevant to my PhD work, and will evaluate them as implementation model for my own operators.

Optimal Query Processing. As already described in Section 2, when splitting visualization processing into special CEP operators or even when using them in the CEP-outbound processing pipeline, they can be optimized, using techniques, such as *Query Optimization* [6, 15] and efficient *Operator Placement* across the different processing nodes of the system [9]. In my PhD work, I want to use prior art in this research field for the optimal processing of my *Visualization* and *Interactivity Operators*.

7. CONCLUSION

This paper illustrated my vision for real-time event stream visualization. When combining CEP with real-time, interactive visualizations, distributed for a large number of mobile, browser-based applications, the streaming system has to solve some major challenges, such as efficient pipeline management, respecting device capabilities, and providing interactive exploration options on the streaming data. From these challenges, I derived three core problems, to be solved in my PhD work, and presented my solution concepts.

To motivate the proposed concepts, the paper includes examples based on real-time visualizations of a financial application, and first results, regarding the continuous rendering performance of web browsers (on mobile devices). In addition, I presented two prospective architectures that I plan to use and evaluate for implementing my solutions.

In the future, I first plan to evaluate multi-query optimization techniques to facilitate reuse of visualization processing pipelines. Thereafter, I will focus on related approaches for CEP and visualization operators and how they can be used to implement my *Interactivity* and *Visualization Operators*. The goal is to run such operators efficiently inside a CEP engine or within a higher level visualization processing pipeline. Eventually, my work will enable the development of highly interactive, mobile analytics tools, working on real-time, streaming data.

8. REFERENCES

[1] Adobe Systems Inc. The NASDAQ Stock Market, Inc. 2009. Online, 06/2012, (tinyurl.com/nasdaq-study).

[2] M. Bostock, V. Ogievetsky, and J. Heer. D^3 Data-Driven Documents. *Visualization & Comp. Graphics, IEEE Trans.*, 17(12):2301–2309, 2011.

[3] E. Chi and J. Riedl. An operator interaction framework for visualization systems. *Symposium on Information Visualization, IEEE*, pages 63–70, 1998.

[4] H. Childs, E. Brugger, K. Bonnell, J. Meredith, M. Miller, B. Whitlock, and N. Max. A contract based system for large data visualization. *Visualization, IEEE*, pages 190–198, 2005.

[5] J. P. Corrigan. Opra updated traffic projections for 2012 and 2013. Technical report, OPRA, 2011. Online, 06/2012, (tinyurl.com/opra-prj).

[6] C. Jin and J. Carbonell. Predicate indexing for incremental multi-query optimization. *Foundations of Intelligent Systems, LNCS*, 4994:339–350, 2008.

[7] E. Kalyvianaki, W. Wiesemann, Q. H. Vu, D. Kuhn, and P. Pietzuch. Sqpr: Stream query planning with reuse. *Proc. ICDE, IEEE*, pages 840 – 851, 2011.

[8] D. Keim, F. Mansmann, J. Schneidewind, J. Thomas, and H. Ziegler. Visual analytics: scope and challenges. *Visual Data Mining, LNCS*, 4404:76–90, 2008.

[9] G. Lakshmanan, Y. Li, and R. Strom. Placement strategies for internet-scale data stream systems. *Internet Computing, IEEE*, 12(6):50–60, 2008.

[10] N. Leavitt. Complex-event processing poised for growth. *Computer, IEEE*, 42(4):17–20, 2009.

[11] J. Lee and G. Grinstein. An architecture for retaining and analyzing visual explorations of databases. *Visualization, IEEE*, pages 101–108, 1995.

[12] Neil McGovern. Introduction to compex event processing in capital markets. Technical report, Sybase, Inc., 2009.

[13] H. Plattner and A. Zeier. *In-Memory Data Management: An Inflection Point For Enterprise Applications*. Springer, 2011.

[14] W. Schroeder and B. Lorenson. *Visualization Toolkit: An Object-Oriented Approach to 3-D Graphics*. Prentice Hall PTR, 1996.

[15] T. Sellis. Multiple-query optimization. *Transactions on Database Systems*, 13(1):23–52, 1988.

[16] B. Shneiderman. Treemaps for space-constrained visualization of hierarchies. *ACM Transactions on Graphics*, 11:92–99, 1998.

[17] Streambase Inc. StreamBase for MMOs, MMOGs, & Social Worlds. Online, 06/2012, (tinyurl.com/sbmmo).

[18] W3C Consortium. HTML5 Working Draft. Online, 06/2012, (dev.w3.org/html5/spec/spec.html).

[19] W3C Consortium. The WebSocket API. Online, 06/2012, (dev.w3.org/html5/websockets).

[20] W. Wang, H. Wang, G. Dai, and H. Wang. Visualization of large hierarchical data by circle packing. In *Proc. SIGCHI*, pages 517–520. ACM, 2006.

[21] E. Wu, Y. Diao, and S. Rizvi. High-performance complex event processing over streams. In *Proc. SIGMOD*, pages 407–418. ACM, 2006.